

---

Bachelorarbeit Medientechnologie

# **Effizientes und realistisches Partikelsystem zur Simulation von Feuer und Rauch in VR- Umgebung**

vorgelegt von

**Miro Steiger**

Erstgutachter: Prof. Dr.-Ing. Arnulph Fuhrmann (Technische Hochschule Köln)

Zweitgutachter: Prof. Dr. rer. nat. Stefan Michael Grünvogel (Technische Hochschule Köln)

Köln, 28.07.2022

Fakultät für  
Informations-, Medien-  
und Elektrotechnik

**Technology  
Arts Sciences  
TH Köln**

# Bachelorarbeit

**Titel:** Effizientes und realistisches Partikelsystem zur Simulation von Feuer und Rauch in VR-Umgebung

**Gutachter:**

- Prof. Dr. Arnulph Fuhrmann (TH Köln)
- Prof. Dr. rer. nat. Stefan Michael Grünvogel (TH Köln)

**Zusammenfassung:** Der Einsatz von Virtual Reality findet in immer mehr Bereichen seinen Nutzen. Im Bereich der Brandbekämpfung könnte die Technik eine sichere und kostengünstigere Alternative zu bestehenden Trainingsmethoden sein. Aktuelle Anwendungen legen bisher jedoch nicht viel Wert auf wirklich immersive Erfahrungen beim Rendering der Brände. Durch realistischere Renderings kann der Nutzer in echte Stresssituationen versetzt werden. Parallax Occlusion Mapping und Raymarching sind zwei Methoden, welche sich für die Darstellung von Feuer und Rauch in Virtueller Realität anbieten. In dieser Arbeit werden daher die beiden vorgeschlagenen Algorithmen in einem Partikelsystem implementiert, verglichen und hinsichtlich ihrer Performance, dem Look und den Einsatzmöglichkeiten in einem Brandsimulator bewertet.

**Stichwörter:** Virtual Reality, Partikelsystem, Parallax Occlusion Mapping, Volumen Rendering, Raymarching, Echtzeitrendering

**Datum:** 28.07.2022

# Bachelors Thesis

**Title:** Efficient and realistic particle system to render fire and smoke in VR

**Reviewers:**

- Prof. Dr. Arnulph Fuhrmann (TH Köln)
- Prof. Dr. rer. nat. Stefan Michael Grünvogel (TH Köln)

**Abstract:** The use of virtual reality extends into more and more areas. In the field of firefighting, the technique could be a safe and cheaper alternative to existing fire training methods. However, current applications focused less on truly immersive experiences when it comes to the rendering the fires. More realistic renderings can put the user in higher stressful situations. Parallax occlusion mapping and raymarching are two methods that are suitable for displaying fire and smoke in a stereoscopic view. In this thesis, the two proposed algorithms are implemented in a particle system, compared and evaluated with regard to their performance, appearance and possible uses in a fire simulator.

**Keywords:** Virtual Reality, Particle System, Parallax Occlusion Mapping, Volume Rendering, Raymarching, Real Time Rendering

**Date:** 28.07.2022

# Inhalt

<b>1 Einleitung</b>	<b>1</b>
1.1 Problem . . . . .	1
1.2 Zielsetzung . . . . .	2
1.3 Struktur der Arbeit . . . . .	2
<b>2 Related Work</b>	<b>2</b>
2.1 Partikelsysteme . . . . .	2
2.2 Parallax Occlusion Mapping . . . . .	3
2.3 Ray Marching . . . . .	3
<b>3 Grundlagen</b>	<b>4</b>
3.1 Virtual Reality . . . . .	4
3.1.1 Konzept . . . . .	4
3.1.2 Tiefenwahrnehmung . . . . .	5
3.1.3 Head-Mounted-Display . . . . .	5
3.2 Feuer- und Rauchsimulationen . . . . .	6
3.2.1 Feuer . . . . .	6
3.2.2 Rauch . . . . .	7
3.2.3 Partikelsysteme . . . . .	7
3.3 Texture Mapping . . . . .	7
3.3.1 Bump Mapping . . . . .	7
3.3.2 Displacement Mapping . . . . .	8
3.3.3 Parallax Mapping . . . . .	9
3.3.4 Parallax Occlusion Mapping . . . . .	9
3.4 Volume Rendering . . . . .	10
3.4.1 Ray Marching . . . . .	11
3.4.2 Volume Ray Marching . . . . .	12
<b>4 Konzeption und Umsetzung</b>	<b>14</b>
4.1 Idee . . . . .	14
4.2 Erstellung der Texturen . . . . .	14
4.2.1 Texture Sheets . . . . .	14
4.2.2 Procedural 3D Noise . . . . .	16
4.3 Shader . . . . .	16
4.3.1 Parallax Occlusion Mapping . . . . .	17
4.3.2 Raymarcher . . . . .	18
4.4 Partikelsystem . . . . .	18
<b>5 Evaluierung der Methoden</b>	<b>19</b>
5.1 Parallax Occlusion Mapping . . . . .	19
5.2 Ray Marching . . . . .	20
<b>6 Ergebnisse</b>	<b>20</b>

6.1	Zusammenfassung . . . . .	20
6.2	Limitationen . . . . .	20
6.3	Ausblick . . . . .	20
	<b>Literaturverzeichnis</b>	<b>21</b>

# 1 Einleitung

Die Simulation von Feuer und Rauch ist ein viel diskutiertes Thema in der Computergrafik. Einerseits spielen diese besonders in Videospielen und Filmproduktionen eine außerordentlich große Rolle. Auf der anderen Seite helfen solche Simulationen auch im Bereich der Gefahrenbekämpfung und -vorbeugung. So kann eine realistische Feuersimulation dazu beitragen, einen Trainingssimulator für die Feuerwehr zu entwerfen [Schlager, 2017]. Der Nutzer kann dabei mithilfe von Head-Mounted-Displays in ein virtuelles Brandszenario versetzt werden, ohne dabei wirklichen Gefahren ausgesetzt zu sein. Eine solche Anwendung findet seinen Nutzen sowohl im Training von Einsatzkräften, als auch bereits bei der Entscheidung einem solchen Beruf nachzugehen. Hierbei gibt es verschiedenste Ansätze um ein künstliches erzeugtes Feuer auf einem Bildschirm anzeigen zu lassen. Eine gängige Methode für das Rendering von Gasen und Flüssigkeiten in Videospielen, in denen sich auch Feuer und Rauch aufgrund ihrer physikalischen Eigenschaften wiederfinden, sind der Einsatz von Partikelsystemen. Die physikalisch korrekte Simulation kann dabei, unter anderem mithilfe von Fluidsimulationen, sehr realitätsnah dargestellt werden. Auf realen physikalischen Eigenschaften basierende Simulationen sind jedoch sehr aufwändig in der Berechnung und bisher kaum für die Echtzeitanwendung gedacht. Gerade in Virtual-Reality-Systemen, in denen die Performance extrem wichtig für das Nutzererlebnis sind, eignet sich die aufwändige Simulation von Fluiden aufgrund ihrer Performance nicht. Als Alternative haben sich hierfür eine Art von Partikelsystemen etabliert, welche sich anstatt der physikalisch korrekten Eigenschaften eher an einer optischen Illusion mithilfe animierter Texturen bedienen. Hierbei ist das Konzept des "Billboardings" ein weit verbreiteter und beliebter Ansatz, um realistischere Renderings der Partikel zu erzeugen. Diese bieten eine optisch überzeugende und dabei noch hocheffiziente Lösung.

## 1.1 Problem

Ein solches Partikelsystem, basierend auf Texturen, kann auf einem flachen Bildschirm realistisch und optisch überzeugend aussehen. In einer VR-Umgebung gerät diese Methode jedoch leider an seine Grenzen. Die Illusion basiert auf der Eigenschaft, dass die flachen Partikel immer zum Nutzer, also der Kamera ausgerichtet sind. Dadurch lässt sich nicht erkennen, dass lediglich flache Texturen zum Einsatz kommen. Die Partikel sehen voluminös aus und täuschen Tiefe vor. In VR-Anwendungen müssen jedoch immer zwei Bilder erzeugt werden, eins für jedes Auge. Dadurch, dass sich die flachen Billboards immer zur jeweiligen Kamera, bzw. zum Mittelpunkt zwischen beiden Augen orientieren, zerstört dies die Illusion und es lässt sich erkennen, dass die Partikel ihre Tiefe lediglich vortäuschen. So sieht ein Feuer-Parikelsystem, basierend auf Texturen, schnell sehr unecht aus und die Immersion ist gestört. Um in einem Traingsszenario der Feuerwehr jedoch einen wirklichen Nutzen zu finden, sollte das Feuer so realistisch wie möglich aussehen. Erst dann wird der Nutzer in eine echte Stresssituation versetzt und kann sich somit besser auf einen Einsatz in der realen Welt vorbereiten.

## 1.2 Zielsetzung

Ziel dieser Arbeit ist es, ein Partikelsystem zu entwickeln, welches alle Vorteile der Billboard-Technik nutzen kann, um auch in Virtual Reality-Umgebung ein realistisches Bild von Feuer und Rauch erzeugen zu können. Durch eine bessere, plausible Darstellung in der Stereo-Ansicht kann beim Nutzer ein realeres Gefühl von Gefahr hervorgerufen werden, welches den Trainingseffekt deutlich erhöhen kann. Dabei soll ein solches Partikelsystem in Unity entworfen werden, welches sowohl in der Stereoansicht funktioniert, als auch in Hinblick auf die benötigte Performance für VR-Renderings die Mindest-Framerate einhalten kann. Hierzu muss ein Shader für die Billboard-Partikel entworfen werden, welcher beide Augen berücksichtigt. Aufgrund der verschiedenen visuellen Eigenschaften und des Verhaltens von Feuer und Rauch müssen jeweils eigene Systeme konzipiert werden.

## 1.3 Struktur der Arbeit

Zunächst wird in Kapitel 2: **Related Work** ein kurzer Überblick über bereits vorhandene Arbeiten und Beiträge aus den Bereichen Partikelsystemen, Parallax Occlusion Mapping, Ray Marching und deren Anwendung in VR gegeben. Das Kapitel 3: **Grundlagen** setzt sich mit den thematischen Grundlagen zu VR, Feuer- und Rauchsimulationen, Mapping Verfahren und Volume Rendering auseinander, welche benötigt werden um die geplanten Partikelsysteme zu implementieren. Im vierten Teil der Arbeit geht es um die **Konzeption und Umsetzung** der gewählten Methoden. Hierbei werden die Schritte von der Idee über die Erstellung der Texturen und Shader bis hin zu fertigen Partikelsystemen beschrieben. Anschließend werden im 5. Kapitel **Evaluierung der Methoden** beide Methoden individuell auf ihre Vor und Nachteile hinsichtlich der Performance und ihrer optischen Qualität geprüft. Im letzten Teil wird in Kapitel 6: **Ergebnisse** eine Zusammenfassung der Ergebnisse, sowie mögliche Anwendungsszenarien der Methoden diskutiert. Abschließen werden einige weitere Optimierungsmöglichkeiten vorgestellt.

## 2 Related Work

Die Idee eines Einsatztrainings für die Brandbekämpfung in Virtual Reality ist nicht neu. Es gibt mit 'Serious Games' sogar eine eigene Kategorie, bei der es sich um Videospiele handelt, bei denen der Bildungsaspekt im Vordergrund steht. Auch in der Brandvorbeugung und -bekämpfung entstanden bereits einige Anwendungen, die versuchen, sich die Möglichkeiten von VR zunutze zu machen. Das Ziel der Anwendungen ist dabei oft das selbe. Sowohl um die Einsatzkräfte in realistischen Szenarios zu trainieren, ohne dabei die physische Gesundheit der Personen aufs Spiel zu setzen [Williams-Bell et al., 2014] , als auch die Umwelt und die finanziellen Mittel zu schonen.

### 2.1 Partikelsysteme

Für die Simulation von Feuer und Rauch werden in computergenerierten Welten aufgrund ihrer Performance seit vielen Jahren überwiegend Partikelsysteme benutzt. Partikelsysteme sind eine kostengünstige Lösung, was die Rechenzeit angeht und eignen sich daher um eben solche

volumetrischen Effekte, die schwer zu modellieren und zu berechnen sind, in Echtzeitsystemen darzustellen zu können. Dies erzeugt auf flachen Bildschirmen die Illusion, dass diese Texturen keine flachen Bilder, sondern voluminös sind. Hierbei handelt es sich um Systeme von einzelnen Partikeln, welche über verschiedene Eigenschaften verfügen und diverse Formen annehmen können. Ein solches System kann Partikel in Form von beispielsweise Punkten, Linien, Sprites oder Meshes emittieren [Reeves, 1983].

In Schlager [2017] wurde bereits mithilfe von Partikelsystemen ein interaktiver Trainingssimulator für die Anwendung eines Feuerlöschers entwickelt. Der Nutzer lernt den korrekten Umgang mit einem Feuerlöscher und muss für die gegebenen Situationen aus verschiedenen Löschmitteln das jeweils am besten geeignete Mittel auswählen und den Brand löschen. Solche Trainings gibt es zwar bereits mit echtem Brand und Feuerlöschern, jedoch lernt der Nutzer hierbei nichts darüber, wie sich ein Feuer in Innenräumen verhält. Auch die Rauchentwicklung im Raum wird hier nicht weiter betrachtet. Der Fokus der Arbeit liegt hier auf der approximiert-realistischen Simulation der Ausbreitung des Feuers, basierend auf Daten des Fire Dynamics Simulators (FDS) [McGrattan und Forney, 2004]. FDS ist eine auf Computational Fluid Dynamics (CFD) basierende Open-Source Software vom National Institute of Standards and Technology (NIST). Brennbare Objekte werden durch ein Voxelgitter repräsentiert, in dem jedes Voxel Informationen wie Temperatur und Brennbarkeit beinhaltet. Bei Schlager liegt der Fokus nicht auf dem realistischen Rendering. In Hinblick auf die Performance kam Schlager zu dem Fazit, dass man einen Kompromiss zwischen realistischem Rendering und realitätsnahem Verhalten des Feuers finden muss.

## 2.2 Parallax Occlusion Mapping

Parallax Occlusion Mapping [Brawley und Tatarchuk, 2004] wurde als Weiterentwicklung des Parallax Mappings [Kaneko et al., 2001] präsentiert. POM wird mittlerweile in vielen Bereichen aufgrund der hohen Details bei gering aufgelösten Meshes [Tatarchuk, 2006] als Alternative zu Vertexdisplacements, bzw. hochauflösenden Geometrien verwendet. Es wurde mit dem Prism Parallax Occlusion Mapping eine Methode vorgestellt um die Silhouetten der Oberflächen korrekt darstellen zu können [Dachsbacher und Tatarchuk, 2007]. Außerdem gibt es bereits Versuche POM im Zusammenhang mit Fluidsimulationen anzuwenden [Kufner, 2017]. Hierbei wurde eine 2D-Fluidsimulation, unter anderem mithilfe von POM, in eine Art Pseudo-Volumen überführt. Der von Kufner festgestellte Nachteil seiner Implementierung ist ein sichtbarer Treppeneffekt, welcher den Nutzen von POM für diese Art der Simulation einschränkt.

## 2.3 Ray Marching

Partikelsysteme, basierend auf Billboards, haben das Problem der Darstellung in VR. Alle Vorteile, die diese Art von Partikelsystem mit sich bringt gehen durch stereoskopisches Rendering verloren, da die Partikel plötzlich flach aussehen oder sich zusammen mit der Bewegung des VR-Headset drehen und kippen können.

Eine Methode um volumetrische Effekte zu rendern ist das Ray Marching. Ray Marching im Zusammenhang mit SDFs gibt es schon seit über 30 Jahren [J. C. Hart et al., 1989]. Ray Marching

bietet neben deutlich realistischeren Renderings den Vorteil, dass dieses auch in der Stereoansicht gut funktioniert und überzeugende Ergebnisse liefert [Wald et al., 2006]. Die Methode bietet jedoch den Nachteil einer aufwändigeren Berechnung und daher auch längeren Renderingzeiten. Es wurde außerdem bereits von Zhang [2020] versucht, die Charakteristiken von volumetrischen Effekten auf ein Billboard-System anzuwenden. Hierbei hängt die Real Time-Performance allerdings stark von der Komplexität der zu rendernden Szene ab.

## 3 Grundlagen

Dieses Kapitel umfasst die technischen Grundlagen auf denen diese Arbeit aufbaut. Es wird ein Einblick in die Funktionsweise von VR gegeben, um das Problem bisheriger Lösungen besser zu verstehen. Außerdem werden bewährte Methoden zur Feuer- und Rauchsimulation gesammelt und erklärt. Ein weiterer Teil dieses Kapitels setzt sich mit grundlegendem Texture Mapping, Bump- und Displacement Mapping, sowie Parallax- und Parallax Occlusion Mapping auseinander. Abschließend werden noch zwei Volume Rendering-Methoden vorgestellt: Ray Marching und Texturbasiertes Volume Rendering.

### 3.1 Virtual Reality

#### 3.1.1 Konzept

Hinter dem Begriff Virtual Reality (VR) verbirgt sich das Konzept einer künstlichen, von Computern generierten Welt. Der Nutzer kann in diese Welt eintauchen und hat dabei die Möglichkeit sich als Betrachter in dieser Welt umzuschauen oder sogar mit dieser Welt zu interagieren. Das erste Konzept eines VR-Headsets mit Kopftracking wurde bereits in den 60er Jahren von Ivan Sutherland entworfen. [Sutherland, 1965, 1968]

Heutzutage gibt es verschiedene Arten von VR. Zum einen die "Non-immersive Virtual Reality". Hierbei steuert der Nutzer seine virtuelle Umgebung, ist sich dabei aber noch bewusst in welcher Realität er sich tatsächlich befindet. Die Interaktion geschieht üblicherweise durch Eingabegeräte wie Controller, Maus oder Tastatur. Ein weit verbreitetes Anwendungsgebiet sind dabei herkömmliche Videospiele. Gegenüberstehend gibt es dagegen die "Fully Immersive Virtual Reality". Hierbei wird der Nutzer durch spezielle Hardware, zum Beispiel mithilfe eines Head-Mounted-Displays (HMD), einem sogenannten VR-Headset, selbst in eine virtuelle dreidimensionale Umgebung versetzt. Durch visuelles, auditives und teilweise auch haptisches Feedback kann der Nutzer dabei immer weiter in die virtuelle Welt eintauchen. Auch hier verfügt der Nutzer über spezielle Eingabegeräte wie dem Headset, Controllern oder Laufbändern. Diese sind jedoch in ihrer Benutzung näher an der bekannten Realität. So kann sich der Nutzer z.B. mit einer Kopfbewegung in der virtuellen Welt umsehen oder Dinge anfassen und mit diesen interagieren. Dieser Einfluss auf die Umgebung sorgt dafür, dass sich eine Simulation echter anfühlen kann.

Die Idee von Fully Immersive Virtual Realities baut dabei darauf auf, die Sinne des Nutzers so überzeugend zu täuschen, sodass dieser glaubt, er befände sich in einer anderen Welt. Die nächste Stufe nach Immersion ist die Präsenz. Präsenz beschreibt hierbei das Gefühl, bzw. die Illusion,

dass sich der Nutzer tatsächlich physisch in dieser computergenerierten Welt befindet und diese nicht mehr von seiner wirklichen Realität unterscheiden kann [Schuemie et al., 2001].

### 3.1.2 Tiefenwahrnehmung

Der Mensch ist ein visuell orientiertes Lebewesen. Daher ist der Einsatz einer VR-Brille einer der wichtigsten Faktoren um eine solche Illusion zu erzeugen. Der Eindruck, sich in einer anderen dreidimensionalen Welt zu befinden, wird von der Illusion von Raum und Tiefe vom Gehirn erzeugt. Die Information dazu werden aus den Bildern beider Augen generiert. Das Gehirn hat einige Möglichkeiten sich damit ein Verständnis des Raumes zu entwickeln. Die Tiefenwahrnehmung wird mithilfe binokularer Disparität erzeugt. Dieser Begriff bezeichnet grob gesagt den kleinen, aber bedeutenden Unterschied zwischen den beiden einzelnen Bildern, welche von den Augen erzeugt werden. Daraus kann das Gehirn etwa abschätzen, wie weit ein Objekt entfernt ist. Diese Disparitäten entstehen durch Informationen wie Verdeckung, Schattenwurf oder die unterschiedlichen Orientierungen von Linien zwischen beiden Blickwinkeln [Tauer, 2010]. Mit Hilfe aller dieser Informationen entsteht ein Eindruck von räumlicher Tiefe, jedoch ist es noch nicht ganz möglich mit den Methoden eine wirklich genaue Einschätzung der Entfernung in dieser Welt zu erhalten. Gerade transparente Objekte lassen sich nicht genau verorten [El Jamy und Marsh, 2019].

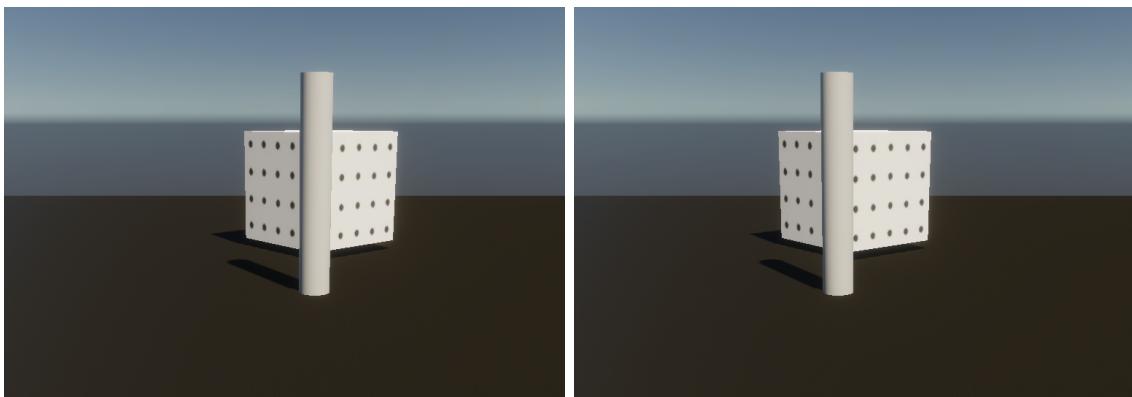


Abbildung 1: Perspektive des linken, sowie des rechten Auges. Die Anzahl der sichtbaren Punkte auf den Seiten des Würfels verdeutlicht die leicht verschiedenen Blickwinkel der Augen.

### 3.1.3 Head-Mounted-Display

Die meisten kommerziellen Systeme basieren heutzutage auf der Nutzung eines HMD. Diese können sowohl Bild, als auch Ton ausgeben. Für diese Arbeit ist jedoch nur die visuelle Komponente interessant. Ein HMD basiert auf zwei Displays, welche sich direkt vor den Augen des Nutzers befinden [Sutherland, 1968]. Diese Displays machen sich die Eigenschaften des menschlichen Sehens zunutze um die Illusion von Tiefe hervorzurufen. Auf jedem Display wird jeweils ein Bild gerendert, welches aus leicht verschobenen Positionen heraus berechnet wird. Dabei werden in der Software, anstatt der üblichen einzelnen Kamera für das Rendering, die Bilder von zwei virtuellen Kameras aufgenommen, welche die Abstände der beiden Augen simulieren [Gateau und Nash, 2010]. Dieser Abstand beträgt den durchschnittlichen Augenabstand eines Menschen. Normalerweise liegt dieser bei ca. 65mm.

Der Einsatz einer VR-Brille bringt einige technische Anforderungen mit sich, welche sich deut-

lich von denen eines herkömmlichen Monitors unterscheiden. Die empfohlenen Spezifikationen unterscheiden sich dabei je nach Art und Hersteller der Brille. Für die Implementierung und das Testing der Methoden dieser Arbeit wurde die HP Reverb G2 mit den folgenden Spezifikationen verwendet.

Bildschirm	2 x 2,89-Zoll-LCD
Auflösung	2160 x 2160 pro Auge 4320 x 2160 kombiniert
Field OF View	~114°
Bildrate	90Hz
Trackingarchitektur	6DoF
Augenabstand	64 mm +/- 4 mm durch Hardware Slider

Tabelle 1: HP Reverb G2 Spezifikationen und Anforderungen [HP Development Company, 2022]

## 3.2 Feuer- und Rauchsimulationen

Chemischen Reaktionen wie Feuer oder Rauch, der als Volumen, bestehend aus winzigen Partikeln beschrieben werden kann, lassen sich nur schwer physikalisch exakt simulieren. Es haben sich hierfür verschiedene Ansätze entwickelt um ein annähernd realistisches Rendering dieser Phänomene zu erschaffen. Es gibt hierbei zum einen die physikalisch basierten Ansätze. Diese beruhen auf aufwändig berechneten Fluidsimulationen und werden vor allem in der Gefahrensimulation verwendet. Diese sind aufgrund ihrer Komplexität aber nur bedingt echtzeitfähig und werden meist zuvor berechnet. Der Fokus liegt dabei meist auf der realistischen Ausbreitung des Feuers und des Rauchs um beispielsweise Fluchtwege zu testen oder Evakuierungsszenarien zu simulieren. Diese Anwendungsfälle müssen nicht in Echtzeit berechnet werden. Daher ist diese Art der Simulation auch für Anwendungen in denen es um die Interaktion mit dem Feuer geht nicht effizient nutzbar. Ein Beispiel dafür ist der Einsatz in einem Simulator, der den Umgang mit dem Feuerlöscher näher bringen soll. Diese basieren zumeist auf effizienteren Partikelsystemen, mit denen sich solche Phänomene durch eine Vielzahl an Parametern ebenfalls darstellen lassen. Dieser Ansatz ist jedoch eher von artistischer Natur. Zwar lassen sich die Partikel, beispielsweise mit Vektorfeldern, genau steuern, um ein realistisches Aussehen zu erzeugen gibt es vor allem in Videospielen andere Methoden, die sich durchgesetzt haben.

### 3.2.1 Feuer

Feuer, bzw. Flammen sind brennende, Licht und Wärme emittierende Gase, erzeugt durch eine chemische Reaktion.

### 3.2.2 Rauch

### 3.2.3 Partikelsysteme

## 3.3 Texture Mapping

Texture Mapping bezeichnet ein Verfahren, welches zweidimensionale Texturen auf ein dreidimensionales Objekt abbildet. Um die flachen Texturen auf die Oberfläche des Meshs abilden zu können, muss das Objekt 'UV-unwrapped' werden. Durch diesen Vorgang wird jedem Punkt auf der Oberfläche des Meshs ein Punkt auf der Textur zugewiesen [Catmull, 1974] [Blinn und Newell, 1976]. Ein sehr einfaches Beispiel zur Veranschaulichung ist dabei das Würfelnetz (Abbildung 2).

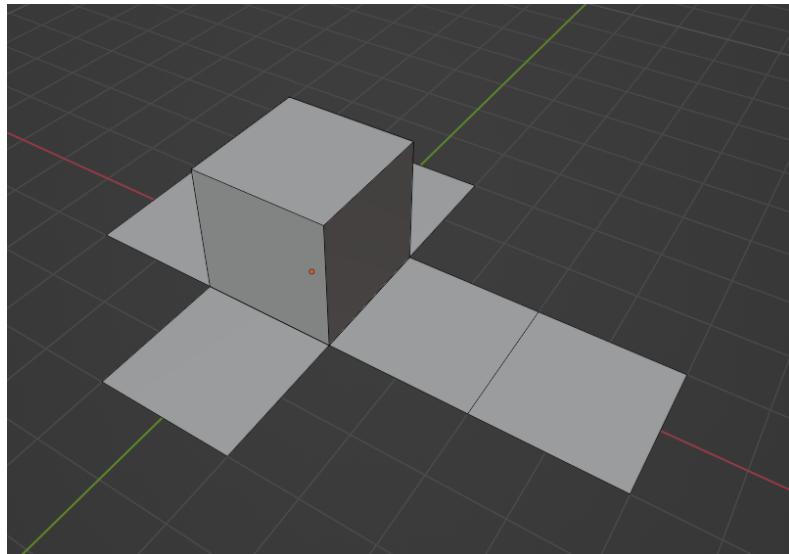


Abbildung 2: Würfelnetz.

Texture Mapping sorgt dafür die Objekte 'anzumalen'. Reale Objekte haben oft sehr detaillierte Oberflächeneigenschaften und sind eigentlich niemals wirklich glatt. Geometrische Unebenheiten und Feinheiten, wie Kratzer, Rillen und Schmutz lassen sich zwar mithilfe von Texturen andeuten, jedoch bleibt die Oberfläche komplett glatt. Diese rauen Oberflächen zu modellieren resultiert aber in einer deutlich höheren Polygon-Anzahl, was die Performance in großen Szenen schnell negativ beeinflussen kann. Daher wurden Mapping-Verfahren als Ergänzung entwickelt um die virtuelle Auflösung solch komplexer Oberflächen kostengünstig zu erhöhen, ohne dabei die Komplexität der Geometrie zu verändern. Dabei gibt es verschiedenste Shader, welche mithilfe weiterer, spezieller Texturen eine deutlich detailliertere Oberfläche simulieren können.

### 3.3.1 Bump Mapping

Eine Möglichkeit, mit der sich die Oberflächen mit mehr Details rendern lassen, ist das Rendering mithilfe von sogenannten Bump Maps. Hierbei werden mithilfe von Texturen, welche zusätzliche Informationen zu den Oberflächen enthalten, Details generiert, welche den Eindruck einer realistischen Struktur der Oberfläche erzeugen. Dabei ist es nicht notwendig, dass die Geometrie an sich Informationen dazu beinhaltet [Blinn, 1978].

Bump Maps sind Texturen, basierend auf Graustufen, bei denen die Helligkeit eines Pixels einen Höhenwert repräsentiert. Schwarz steht mit dem Wert 0 für den tiefsten Punkt und weiß mit dem Wert 1 für den höchsten Punkt. Eine verbesserte Variante der Bump Maps sind die Normal Maps. Hier werden die Richtungen der Normalen in jedem Pixel durch einen Vektor repräsentiert, welcher sich aus den RGB-Werten eines jeden Pixels ergibt. Daraus lassen sich Schattierungen simulieren, welche kleine Unebenheiten (mit geringer Tiefe) wie Beulen oder Kratzer realistischer aussehen lassen. Dieser wird aus Lichtquellen, deren Einfallrichtung und den Normalen aus der Textur berechnet. Somit wird auch bei geringer Polygonanzahl eine deutlich realistischer aussehende Oberfläche gerendert. [Cohen et al., 1998].

Diese Texturen bieten einen sehr kostengünstigen Ansatz um Tiefe zu simulieren. Shader basierend auf diesen Methoden sind dabei aber stark blickwinkelabhängig und sehen schnell unnatürlich verzerrt aus. Von vorne betrachtet funktioniert die Illusion, je spitzer jedoch der Winkel zwischen Betrachter und Oberfläche wird, desto auffälliger wird die Tatsache, dass die Silhouette des Objekts immer noch flach ist, da die Geometrie hierbei nicht verändert wird. Zusätzlich wird die Effektivität von reinem Normal Mapping in VR-Umgebungen aufgrund der Unabhängigkeit vom Betrachtungswinkel geschwächt.

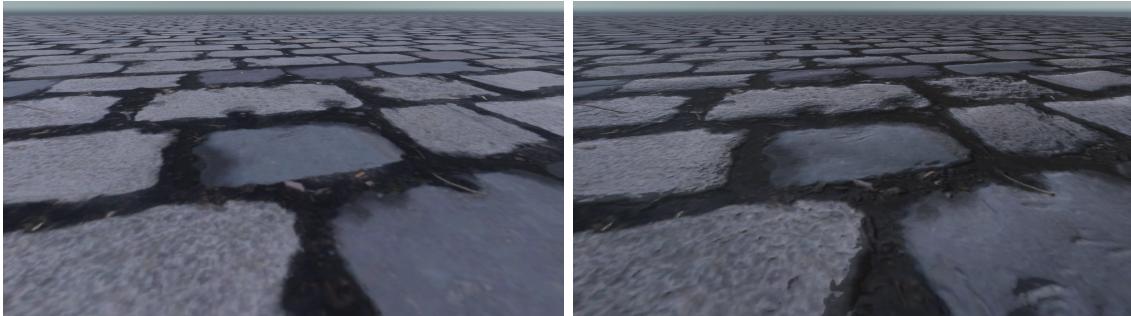


Abbildung 3: links: Einfache diffuse Beleuchtung der Textur, rechts: Normal Mapping. Beide Geometrien sind komplett flach. Hierbei sieht man gut den Unterschied zwischen dem flachen Look der diffus beleuchteten Textur und dem realistischeren Aussehen durch Einbeziehen der Oberflächennormalen aus der Normal Map.

### 3.3.2 Displacement Mapping

Mit Displacement Mapping werden dagegen tatsächlich mithilfe von Heightmaps die Positionen der Vertices entlang ihrer Normalen versetzt [Cook, 1984; Cook et al., 1987]. Dadurch kommt es nicht zu blickwinkelabhängigen Artefakten und die Illusion von Tiefe wird real. Objekte sehen aus einem flachen Winkel betrachtet nicht mehr glatt aus, sondern haben tatsächlich Struktur in ihrer Oberfläche. Damit dieser Effekt jedoch zustande kommt, muss das Mesh in einer gewissen Auflösung zur Verfügung stehen. Je nach Detailreichtum der Heightmap muss die Geometrie dabei in weitere Polygone unterteilt werden. Der Vorteil hierbei ist der hohe Grad an Realismus. Ein deutlicher Nachteil liegt dabei allerdings in der Performance. Ein weitgehender Einsatz von Displacement Mapping kann eine hohe Polygonanzahl schnell negativen Einfluss auf die Renderingzeiten nehmen.

### 3.3.3 Parallax Mapping

Parallax Mapping (oder auch Offset (Bump-)Mapping) ist eine weiter Methode, um sich die Möglichkeiten von Bump Mapping-Verfahren zu nutze zu machen. Anders als bei tatsächlicher Modifizierung der Vertices durch Displacement Maps werden hier nur die Texturkoordinaten abhängig vom Blickwinkel verschoben. [Kaneko et al., 2001; Welsh, 2004] Durch Bewegung der Oberfläche oder des Betrachters entsteht somit ein realistischerer Eindruck von Tiefe in der Textur, welcher den des Displacement Mappings approximiert darstellt. Dabei ist Parallax Mapping allerdings immer noch deutlich effizienter als echtes Vertex-Displacement und eignet sich daher eher für Echtzeitrenderings. Parallax Mapping alleine simuliert zwar den Parallax-Effekt, jedoch ist es hiermit nicht möglich die Silhouette zu verändern und Selbstschattierung oder -verdeckung vorzutäuschen.

### 3.3.4 Parallax Occlusion Mapping

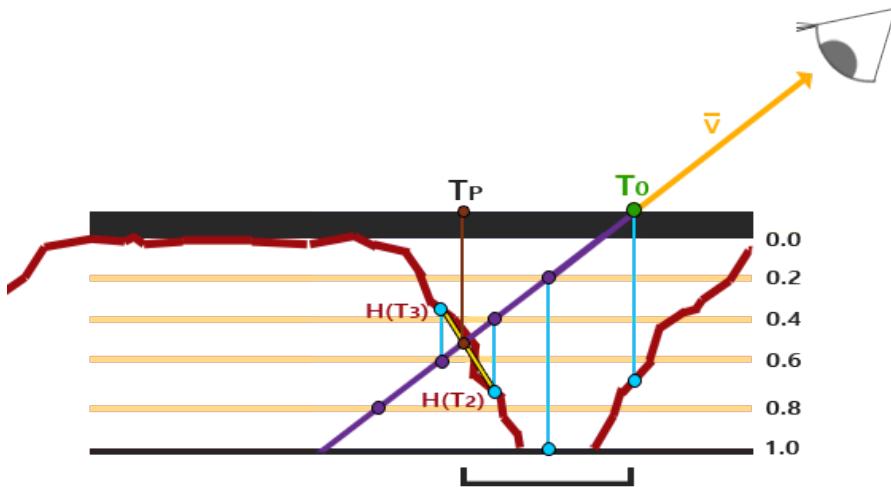


Abbildung 4: Verschiebung der Texturkoordinaten beim Parallax Occlusion Mapping

Parallax Occlusion Mapping (POM) ist eine komplexere, verbesserte Variante des Parallax Mapping, basierend auf einem Per-Pixel Ray Tracing. Das Ziel ist auch hier wieder das vorherige: Detaillierte Oberflächen ohne den Preis von teuren Vertexverschiebungen. Diese Details lassen sich abhängig vom Blickwinkel aus jeder Perspektive korrekt darstellen. Mithilfe von dynamischer Beleuchtung und Selbstokklusion kann außerdem eine korrekte Selbstschattierung simuliert werden [Brawley und Tatarchuk, 2004; Tatarchuk, 2006]. Die Informationen aus der Heightmap werden hierbei für Berechnungen im Fragmentshader verwendet. Anstatt wie beim Displacement Mapping Details zu extrudieren wird bei POM in die Tiefe simuliert. Hierbei wird die Oberfläche der Wert 1.0, und damit der höchste Punkt, angenommen. Zunächst wird für jeden zu rendernden Pixel mittels Ray Castings vom Betrachter zur Geometrie-Oberfläche ein Schnittpunkt  $T_0$  ausfindig gemacht. Der Strahl  $\vec{v}$  wird nun weiter durch das, durch die Heightmap ausgedrückte, Volumen verfolgt und Schrittweise gesampled. Wenn sich der Strahl mit der Heightmap im Punkt  $T_P$  schneidet kann mit der Position die neue zu rendernde Texturkoordinate ermittelt werden. Je geringer die Schrittweite ist, desto genauer kann die Oberfläche gerendert werden. Ist die Schrittgröße zu groß kann die Heightmap nicht detailliert genug abgetastet werden und es entstehen Aliaseffekte, es zeichnet sich ein Treppeneffekt ab (Abbildung 6).

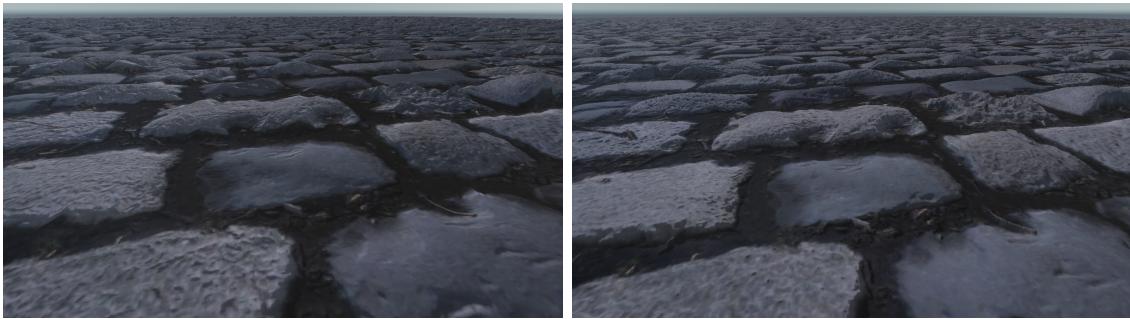


Abbildung 5: links: Displacement Map, Normal Map und 50-facher Tesselation, rechts: Parallax Occlusion Mapping mit 64 Samples. Beide Methoden sehen sich sehr ähnlich, der Unterschied ist jedoch, dass für die Displacement Map-Methode in diesem Beispiel die 50-fache Anzahl an Dreiecken gebraucht wird.



Abbildung 6: Aliasingeffekt bei zu geringer Sampleanzahl mit 4 Schritten

### 3.4 Volume Rendering

Unter dem Begriff Volume Rendering versteht sich eine Reihe von Methoden, die es ermöglichen ein 3D-Datenset zu visualisieren. Anders als beim Rendern von Geometrien mit einer festen Oberfläche, geht es hierbei darum alle Daten aus dem jeweiligen Volumen darstellen zu können. Dies sind beispielsweise volumetrische Effekte wie Feuer und Rauch, Wolken oder Nebel, welche sich, aufgrund ihrer gasförmigen Eigenschaften, nicht wirklich realistisch mit Geometrie darstellen lassen. Volumen haben keine wirkliche Oberfläche, sondern besteht aus unzähligen winzigen Rußpartikeln. Möchte man also solche Volumen darstellen, so geraten die bisherigen Herangehensweisen, die sich die Oberflächen der Objekte zunutze machen, schnell an ihre Grenzen. Beim Grundgedanken des Volume Renderings geht es daher um die Frage, wie das Licht durch diese Volumen wandert und welchen Einfluss die Partikel darauf haben. Es gibt einige Faktoren, die beeinflussen was das Auge am Ende von diesem Volumen sieht. Wenn das Licht durch solch ein Volumen wandert kann es absorbiert, reflektiert oder gestreut werden. Die Abschwächung

durch Streuung und Absorption des Lichtes beim Durchlaufen eines Volumens kann vereinfacht mithilfe des Lambert-beerschen Gesetzes wie in Gleichung 3.1 approximiert werden [Mayerhöfer et al., 2020]. Der Absorptionskoeffizient  $\sigma_a$  beschreibt dabei die Wahrscheinlichkeit, dass ein Photon auf der Strecke  $d$  durch das Medium absorbiert oder anderweitig reflektiert wird.

$$I_{out} = I_{in} \cdot e^{-(d \cdot \sigma_a)}. \quad (3.1)$$

### 3.4.1 Ray Marching

Ray Marching ist eine dieser Möglichkeiten ein Volumen darzustellen. Die Idee hierbei ist es – ähnlich wie beim Ray Tracing – Strahlen von der Kamera aus durch jeden Pixel des zu rendernden Bildes zu schicken und dabei zu prüfen, ob der Strahl auf etwas trifft. Der Unterschied besteht hierbei jedoch darin, dass sich Ray Marching auf sogenannten '*signed distance functions*' (SDF) beruht. SDFs beschreiben, wie weit die nächstgelegene Oberfläche entfernt ist. Befindet sich der Strahl noch außerhalb der Geometrie, so ist der Wert der Funktion positiv, befindet sich der Strahl innerhalb, so wird der Wert negativ. Dazu wird die Entfernung zu allen Objekten in der Szene berechnet.

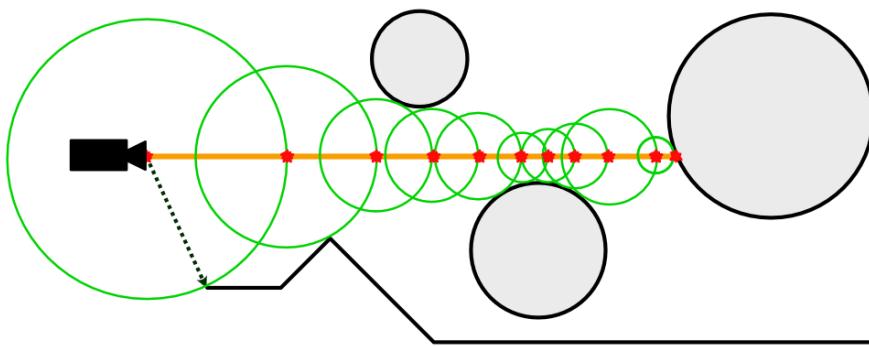


Abbildung 7: Sphere Tracing: Der Algorithmus wird so lange wiederholt, bis der Radius/Abstand zum nächstgelegenen Objekt unter einen bestimmten Grenzwert fällt.

Im ersten Schritt muss also die Distanz zur nächstgelegenen Oberfläche gefunden werden. Daraus ergibt sich ein Radius in dem sich mit Sicherheit nichts anderes befindet. Der Strahl bewegt sich nun um die Länge des berechneten Radius' in seine vorgesehene Richtung. Anschließend werden diese beiden ersten Schritte an der neuen Position wiederholt. Dies passiert so lange, bis die Entfernung gegen einen festgelegten Grenzwert geht. Dies deutet darauf hin, dass eine Oberfläche getroffen wurde. Diese Variante von Ray Marching wird auch '*Sphere Tracing*' genannt [J. Hart, 1996]. Sphere Tracing ist, im Gegensatz zu einer konstanten Schrittweite, ein effizienter Algorithmus um Oberflächen entlang des Strahls zu finden, da hiermit die Anzahl der benötigten Schritte deutlich reduziert werden kann.

Jede Oberfläche einer Szene kann durch eine solche Distanzfunktion dargestellt werden. Eine Kugel lässt sich also beispielsweise durch die folgende Funktion mit ihrer Position in Weltkoordinaten und einem Radius darstellen:

```
float sdfSphere( vec3 pos, float radius ){
```

```

    return length(pos)-radius;
}

```

Listing 1: SDF einer Kugel im Ursprung

Mithilfe dieser Funktionen lassen sich mehrere SDFs auf verschiedene Art und Weise kombinieren und neue Formen erzeugen (Abbildung 8). Dabei gibt es Möglichkeiten wie beispielsweise die Vereinigung zweier Objekte, die Schnittmenge oder auch die Differenz. Mit dem Smooth Union-Operator lassen sich beispielsweise sogenannte Blobs oder Metaballs erzeugen, indem zwischen den Oberflächen interpoliert wird, sodass diese Geometrien den Anschein erwecken miteinander zu verschmelzen.

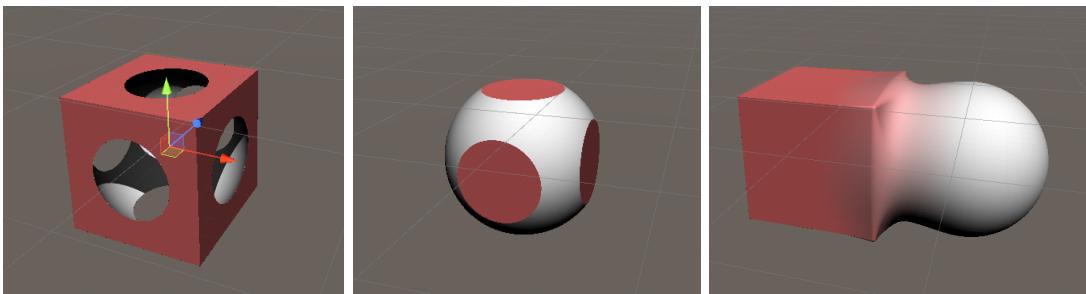


Abbildung 8: Beispiele von verschiedenen SDF-Operatoren. Von links: Subtraction, Intersection, Smooth Union

### 3.4.2 Volume Ray Marching

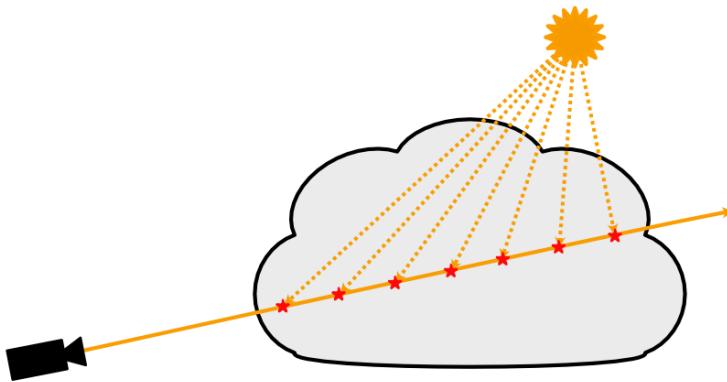


Abbildung 9: Vereinfachte Darstellung des Volume Ray Marching. Der Weg des Lichtes durch das Medium in der Realität aufgrund von In- und Out-Scattering, Absorption oder Emissionen innerhalb des Volumens komplexer.

Möchte man nun aber komplexere Volumen darstellen, so gelingt dies mithilfe einer Volumentextur. Der Ansatz bleibt dabei der Gleiche. Es wird ein Objekt benötigt, welches als eine Domain fungiert. Dieses kann in Form von oben beschriebenen SDFs repräsentiert sein. Innerhalb dieser Geometrie befindet kann nun alles mögliche dargestellt werden, da hierbei keine Geometrie die Grundlage des Volumens darstellt. Es lässt sich mittels Code beispielsweise innerhalb eines Würfels eine Kugel darstellen, sodass es für den Betrachter aussieht wie eine wirkliche Kugel.

Wurde per Sphere Tracing eine Oberfläche ermittelt, so wird in festgelegten Schritten (sample rate) das Innere der Geometrie durchlaufen und an jedem Schritt das ein Wert berechnet, der sich

auch Farbe, Dichte und Tiefe im Volumen berechnet (Abbildung 9). Mithilfe von Gleichung 3.2 [Drebin et al., 1988] lässt sich durch das aus einem Voxel austretende Licht ein Farbwert bestimmen. Voxel sind das Äquivalent zu einem Pixel im dreidimensionalen Raum und wird durch ein 3D-Gitternetz repräsentiert.

$$C_{out} = C_{in} \cdot (1 - O(x)) + C(x) * O(x). \quad (3.2)$$

«<Formel stimmt nicht ganz, muss ich noch mal recherchieren»>

## 4 Konzeption und Umsetzung

### 4.1 Idee

Da die Ergebnisse dieser Arbeit relevant für das KoViTReK Forschungsprojekt<sup>1</sup> sein könnten und dieses in der Laufzeit- und Entwicklungsumgebung Unity<sup>2</sup> implementiert werden soll, werden auch die Partikelsysteme in der selben Umgebung entworfen.

Ein wichtiger Faktor bei der Darstellung von Rauch ist die Beleuchtung. Eigenschaften wie Streuungen und Reflexionen innerhalb des Mediums zu simulieren ist aufwändig zu berechnen und bisher nicht für die Echtzeitberechnung in VR-Systemen, welche deutlich höhere Anforderungen an die Performance der Anwendung haben, gedacht. Der erste Ansatz basiert auf einer Idee von Mederic Chasse, Technical Art Director bei Highwire Games. Dieser schlägt einen Workflow vor, mit dem aufwändige Berechnungen solcher Simulationen in einer VFX Software erstellt, berechnet und gerendert werden. Die Animation wird in einem Texture Sheet gespeichert und kann somit den Einsatz in einer Game Engine sehr effizient machen und eine realistische Darstellung bieten [Chasse, 2018]. Da der Fokus dieser Arbeit nicht auf der Fluidsimulation liegt und das Setup der Simulation mithilfe von Anleitungen und Tipps artistischer Natur aus Internetforen erzeugt wird, wird an dieser Stelle nicht weiter darauf eingegangen, wie die VFX-Software den Rauch berechnet und erzeugt. Stattdessen wird in **Kapitel 4.2.1** der Workflow beschrieben und auf einige Parameter der Simulation eingegangen.

Für die Darstellung mithilfe von Ray Marching werden richtige Volumendaten, bzw. 3D-Texturen benötigt. Diese werden innerhalb von Unity mithilfe eines Shaders prozedural erzeugt. tba

### 4.2 Erstellung der Texturen

#### 4.2.1 Texture Sheets

Die Texturen werden auf zwei verschiedene Arten erstellt. Für die Variante mit Parallax Occlusion Mapping werden zwei Texturen in der von SideFX entwickelten 3D-VFX-Grafiksoftware Houdini<sup>3</sup> erstellt. Houdini bietet die Möglichkeit relativ einfach und schnell physikalisch basierte und visuell überzeugende Rauch- und Feuersimulationen zu erstellen. Die Simulationen basieren auf Fluidberechnungen, wodurch ein realistisches Verhalten erzeugt und durch eine Vielzahl an Parametern beeinflusst werden kann.

Hier noch kurz die wichtigsten Parameter beschreiben, z.B. Buoyancy, Dichte, Short tail des Rauchs, damit dieser eher aussieht wie ein Rauchball

Dabei wird nicht nur die Beleuchtung der Oberfläche aus verschiedenen Richtungen, sondern auch die Streuung, Reflexion und Absorption des Lichts innerhalb des Volumens berechnet und simuliert. Eine Möglichkeit die Houdini bietet, ist das Rendering der Animation in verschiedene Layers. So kann jedes einzelne Licht separat berechnet, gerendert und hinterher separat verwendet werden. Dazu werden Lichtquellen wie in **Abbildung 10** aus allen sechs Richtungen zum

---

<sup>1</sup>[https://www.th-koeln.de/anlagen-energie-und-maschinensysteme/kovitrek\\_87259.php](https://www.th-koeln.de/anlagen-energie-und-maschinensysteme/kovitrek_87259.php)

<sup>2</sup><https://unity.com>

<sup>3</sup><https://www.sidefx.com/products/houdini/>

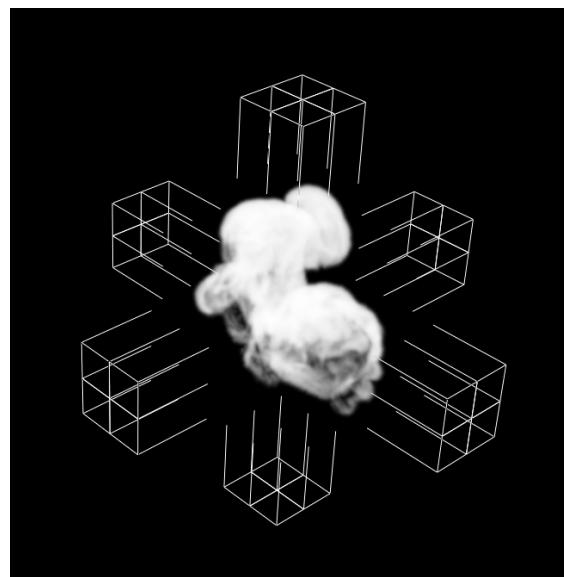


Abbildung 10: Setup der Beleuchtung der Rauchsimulation in Houdini.

Rauch hin angeordnet, sodass jeweils eine Seite beleuchtet wird. Die Animation des Rauchs wird daraufhin wie in **Abbildung 11** in eine sich selbst nahtlos wiederholende Animation geschnitten und als einzelne Frames gerendert. Dazu muss im ersten Schritt eine Animation mit der doppelten Anzahl an Frames erzeugt werden, damit diese in der Hälfte geschnitten werden kann. Nun werden die beiden Abschnitte übereinander gelegt, sodass sich die Schnittkante auf gegenüberliegenden Seiten befinden. Somit sind der erste und der letzte Frame nahezu identisch. Als letztes wird ein Crossfade über beide Abschnitte gelegt, sodass die Clips jeweils an der Schnittkante voll eingeblendet sind. Somit ist der Schnitt nicht mehr sichtbar und das Video lässt sich unendlich wiederholen. Die Vorteile von dieser Methode werden in **Kapitel 5.1** noch weiter erläutert. Um die verschiedenen Texturlayer in Unity verwenden zu können werden die Lichtrichtungen jedes einzelnen Frames in jeweils einen RGBA-Channel gelegt. So kann die Ansicht des Rauchs, in der beispielsweise nur die linke Seite beleuchtet wird in den Roten Kanal einer Textur gespeichert werden. Dies passiert ebenfalls für die anderen Richtungen von links, unten und oben. Die Ergebnisse sind in **Abbildung 12** zu sehen.

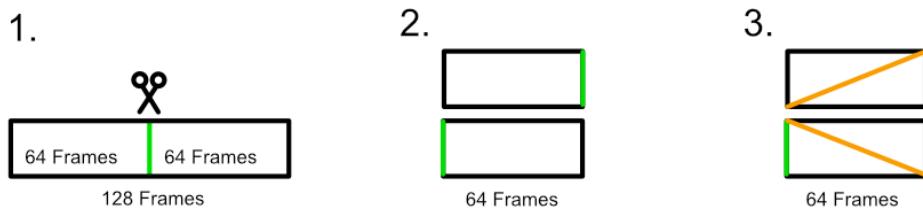


Abbildung 11: Schritte um Loop einer Rauchanimation ohne sichtbaren Schnitt zu erzeugen.

Die verschiedenen Farbkanäle können nun wieder in eine gemeinsame Textur zusammengeführt werden um dann in Unity in einem Shader weiterverarbeitet zu werden. Dabei wird jeder Frame mit einer Auflösung von 128x128 Pixel gespeichert. Bei einer Frameanzahl von  $8 \times 8 = 64$  Frames entspricht das also einer Auflösung der ganzen Textur von  $1024 \times 1024$  Pixel.

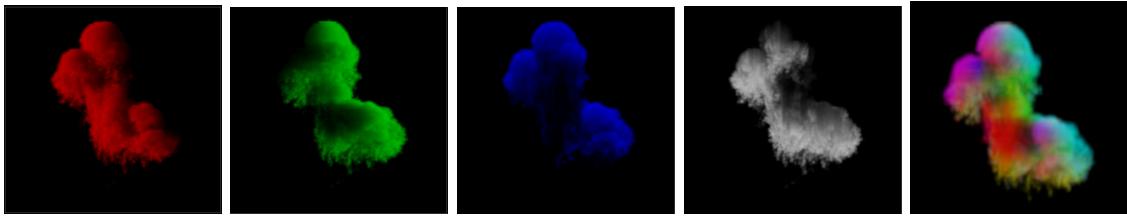


Abbildung 12: Ein einzelner Frame der Simulation in seine Farbkanäle aufgeteilt. Die Richtungen aus denen der Rauch jeweils beleuchtet wurde ist dabei gut zu erkennen. Von links: Roter Kanal(links), Grüner Kanal(rechts), Blauer Kanal(oben), Alpha Kanal(unten), Zusammengeführte Textur.

Damit Parallax Occlusion Mapping angewendet werden kann werden jedoch zusätzlich Informationen über die Höhe benötigt. Dazu wird eine zweite Textur mit den gleichen Schritten wie zuvor erzeugt. Neben der Höheninformation wird außerdem noch die Alpha-Werte, sowie zwei weitere Lichtrichtungen – von vorne und von hinten – in die verschiedenen Kanäle gespeichert.

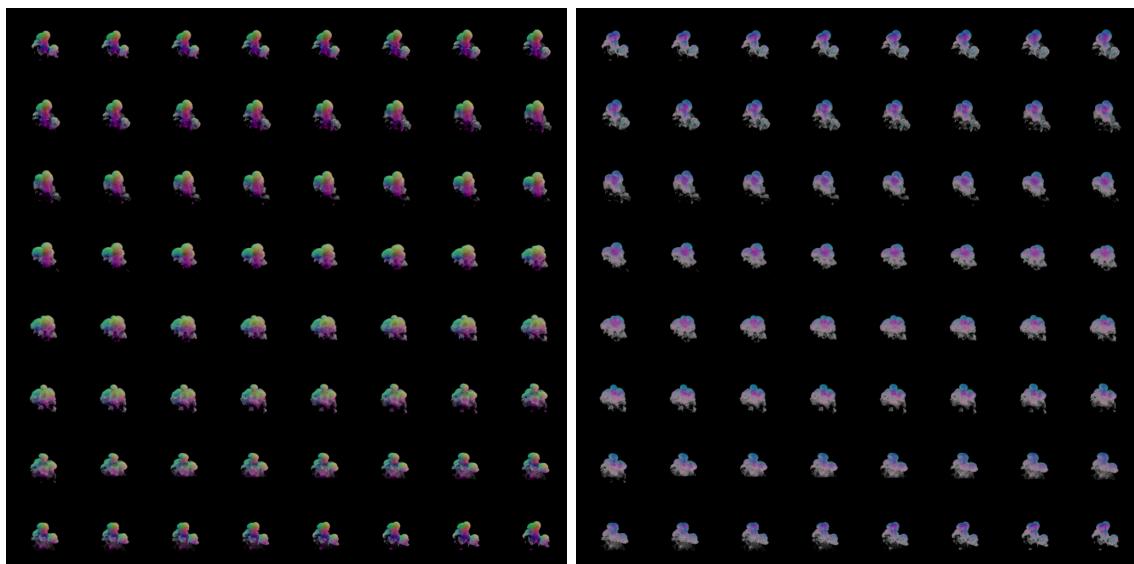


Abbildung 13: Exportierte Spritesheets aus der Rauchsimulation in Houdini.

#### 4.2.2 Procedural 3D Noise

TODO

### 4.3 Shader

Für das Rendering der Partikel werden zwei Shader entwickelt. Diese basieren auf Parallax Occlusion Mapping und auf einem Raymarchingalgorithmus durch eine Volumentextur. Für POM wird Unitys Shader Graph verwendet. Shader Graph ist, genau wie Unitys VFX Graph, ein Node-Editor um schnell und übersichtlich Shader zu entwickeln. Hier besteht die Möglichkeit auch eigene Funktionen in Nodes zu verpacken und somit auf visuelle Art und Weise zu entwickeln. Dies macht vor allem das Debugging einfacher und erlaubt das schnelle Prototyping von Ideen, da einiges an Boilerplate-Code erspart wird.

### 4.3.1 Parallax Occlusion Mapping

Um die einzelnen Frames aus den Texture Sheets als Animation darstellen zu können, wird ein Flipbook-Node<sup>4</sup> verwendet. Dieser manipuliert die UV-Koordinaten mithilfe der Information über Zeilen und Spalten des Texture Sheets, sodass die Koordinaten eines jeden Frames abhängig von der Zeit ausgegeben werden. Somit lässt sich auch die Framerate anpassen, mit der die Animation abgespielt werden soll.

Diese UV-Koordinaten werden nun in einen Parallax Occlusion Mapping-Node weitergegeben, hier wird ebenfalls der Node von Unity<sup>5</sup> benutzt. Dieser berechnet, basierend auf den übergebenen UV-Koordinaten des Flipbook-Players, sowie einer Heightmap und dem Blickwinkel die anzuzeigenden Texturkoordinaten. Es lassen sich außerdem die Anzahl der Schritte mit denen die Heightmap abgetastet werden soll, sowie die Amplitude des Offsets festlegen. Für die Implementierung des Rauchs wurde die Schrittanzahl auf 150 und die Amplitude auf 1 bis 3.5 gesetzt. Diese Werte haben sich durch ausprobieren ergeben. Eine höhere Amplitude verzerrt den Rauch so stark, sodass die in Kapitel 3.3.4 beschriebenen Artefakte auftreten.

#### »SCREENSHOT VON ZU STARK VERZERRTEM RAUCH«

Da die Beleuchtung des Rauchs zuvor in Houdini durchsimuliert und abgespeichert wurde muss im Shader nun eine Logik implementiert werden, welche jeweils die richtige Beleuchtung, abhängig von der Lichtrichtung anzeigt. Die durch den Flipbook- und POM-Node modifizierten UV-Koordinaten werden nun auf das Texture-Sheet abgebildet und somit die Animation durchlaufen. Da durch das Parallax Mapping die Koordinaten je nach Amplitude unterschiedlich stark verzerrt werden, muss der Rauch maskiert werden um das Fragmente, die am Rand erscheinen auszublenden. Mit der maskierten Animation kann dann unter einbeziehen des Main Lights (in Unity definiert als die Sonne, also ein directional light). Directional Lights haben keine Position, lediglich eine Richtung in die sie zeigen. Mithilfe dieser Richtung kann die korrekte Lightmap des Rauchs ausgewählt werden oder je nach Richtung des Lichts zwischen verschiedenen Maps interpoliert werden. Im Prinzip wird also für jede Achse des Lichts abgefragt, in welche Richtung das Licht zeigt und der entsprechende Farbchannel der Rauchtextur zurückgegeben.

```
if(direction.X > 0){
    return smokeLightmap_T1.R;
}
else {
    return smokeLightmap_T2.R; //Licht von links
}
```

Listing 2: Logik zur Auswahl der richtigen Lightmap am Beispiel der X-Richtung des Lichts

<sup>4</sup><https://docs.unity3d.com/Packages/com.unity.shadergraph@12.0/manual/Flipbook-Node.html>

<sup>5</sup><https://docs.unity3d.com/Packages/com.unity.shadergraph@12.0/manual/Parallax-Occlusion-Mapping-Node.html>

### 4.3.2 Raymarcher

Da Feuer und Rauch volumetrische Effekte ohne feste Oberfläche sind, lassen sich diese nicht wirklich durch Geometrie abbilden. Die Idee ist also ein Volumen zu erzeugen und Strahlen durch dieses Volumen zu schicken, welches in festen Abständen das Volumen abtastet um daraus Informationen zu erhalten. Hierbei wird das Licht, welches ebenfalls in dieses transluzente Volumen eindringt und absorbiert oder gebrochen wird, an jedem Samplepunkt berechnet.

Die Texturen werden daher wie in **Kapitel 4.1** beschrieben durch generiertes Rauschen erzeugt.

## 4.4 Partikelsystem

Um die Partikelsysteme umzusetzen wird der relativ junge, von Unity entwickelte Editor 'Visual Effects Graph' (kurz: VFX Graph) verwendet. VFX Graph ist ein nodesbasierter Editor, um schnell dynamische und komplexe Partikelsysteme zu erzeugen<sup>6</sup>. Im Gegensatz zum älteren Shuriken-Partikelsystem von Unity werden die Partikel hier auf der GPU simuliert, wodurch das System deutlich an Performance gewinnt und ein mehr Partikel zeichnen kann. Shuriken nimmt die Berechnungen im Gegensatz zum VFX Graph auf der CPU vor<sup>7</sup>. Gerade für VR-Anwendungen bietet sich also dieses neue System an. VFX-Graph hat jedoch nur sehr begrenzte Möglichkeiten, was Physiksimulationen und Kollisionen der Partikel angeht. Es muss also ein System erstellt werden, welches trotz der Einschränkungen ein möglichst realistisches Verhalten der Feuer- und Rauchpartikel gewährleistet.

---

<sup>6</sup><https://unity.com/de/visual-effect-graph>

<sup>7</sup><https://docs.unity3d.com/Manual/ChoosingYourParticleSystem.html>

## 5 Evaluierung der Methoden

Beide Methoden haben ihre Vor- und Nachteile. Die realistische Darstellung von voluminösen Materialien verlangt eine besondere Herangehensweise, welche sich – im Gegensatz zu undurchsichtigen, festen Oberflächen – nicht überzeugend mit herkömmlichen Texture-Mapping Methoden umsetzen lässt. Die Implementierung der Systeme ist aufgrund der begrenzten Zeit im Rahmen dieser Arbeit nur prototypisch mithilfe der zur Verfügung stehenden Funktionen aus Unity implementiert und daher nicht bestmöglich optimiert. In Kapitel 6.3 werden einige Optimierungsmöglichkeiten präsentiert. Die Vor- und Nachteile, sowie das Potential für den Nutzen der jeweiligen Methoden lassen sich dennoch aus dem Entwurf herausarbeiten und werden im Folgenden beschrieben.

### 5.1 Parallax Occlusion Mapping

PRO:

- Fluidsimulation -> geiler Look
- Beleuchtung ist recht einfach zu machen
- Beleuchtung ist dynamisch für eine geringe Zahl an Lichtquellen - Für dunklen Rauch ist die genaue Berechnung von Schatten nicht so wichtig - In Texturen lassen sich noch mehr Infos speichern (Color-Key (für Feuer), Emission, Gradients, etc.)
- Geringe Shaderlaufzeit
- Unendlich wiederholbar
- Sieht von weiter weg ganz gut aus
- Offset wird für beide Augen individuell berechnet
- Startframe kann zufällig gewählt werden -> gibt dem Rauch ein bisschen Variation ohne Overhead

POM ist eine komplexere Variante des Normalmappings. Dementsprechend ist auch POM eine Technik für feste Oberflächen und weniger für den Einsatz bei der Simulation von Volumen gedacht. Um aber die Effizienz der billboard-basierten Partikelsysteme nutzen zu können ist das Parallax Occlusion Mapping eine interessante Möglichkeit. Trotz der erhöhten Komplexität des Algorithmus gegenüber klassischeren Mappingverfahren läuft die Anwendung auf dem Testsystem stabil mit 90FPS. Hier gibt es also keine Bedenken hinsichtlich der Shaderperformance. Auch die aufwändige Berechnung der Fluidsimulationen wird hier im Voraus erledigt. Somit ist es möglich realistisch aussehende Partikelsysteme unter geringem Rechenaufwand in der Echtzeitanwendung zu erzeugen.

CONTRA

- Transparenz ist irgendwie weird
- Geringe Auflösung der einzelnen Frames
- > Houdini Non-Commercial-Version erlaubt keine höhere Auflösung
- Basiert immernoch auf Billboards
- > Trotz simulierter Tiefe ein flacher Eindruck in VR
- Funktioniert nicht gut bei mehreren Lichtquellen
- Schattenwurf auf Umgebung funktioniert nicht

- Beleuchtung der Umgebung auch nicht möglich

## 5.2 Ray Marching

PRO:

- Sehr günstige AO aufgrund der bereits berechneten Distanz
- Lichtstrahl in Volumen weiterverfolgen kann Schatten erzeugen
- 

CONTRA:

- Aufwändiger Berechnung
- 3D-Datenset beinhaltet keine Animation der Textur

# 6 Ergebnisse

Hier kommen die Erkenntnisse meiner Arbeit rein

## 6.1 Zusammenfassung

## 6.2 Limitationen

## 6.3 Ausblick

## Literatur

- Blinn, James (1978). „Simulation of wrinkled surfaces“. In: *Proceedings of the 5th annual conference on Computer graphics and interactive techniques - SIGGRAPH '78*. ACM Press, S. 286–292. DOI: 10.1145/800248.507101.
- Blinn, James und Martin E. Newell (Okt. 1976). „Texture and Reflection in Computer Generated Images“. In: *Commun. ACM* 19.10, S. 542–547. ISSN: 0001-0782. DOI: 10.1145/360349.360353.
- Brawley, Zoe und Natalya Tatarchuk (2004). „Parallax occlusion mapping: Self-shadowing, perspective-correct bump mapping using reverse height map tracing“. In: *ShaderX<sup>3</sup>: Advanced Rendering with DirectX and OpenGL*. Hrsg. von W. Engel. Charles River Media, S. 135–154.
- Catmull, Edwin (1974). „Computer display of curved surfaces“. Diss., S. 35–41. DOI: 10.1145/280811.280920.
- Chasse, Mederic (2018). *Smoke Lighting and texture re-usability in Skull & Bones*. URL: <https://realtimevfx.com/t/smoke-lighting-and-texture-re-usability-in-skull-bones/5339/24> (besucht am 20.07.2022).
- Cohen, Jonathan, Marc Olano und Dinesh Manocha (1998). „Appearance-Preserving Simplification“. In: *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '98. Association for Computing Machinery, S. 115–122. DOI: 10.1145/280814.280832.
- Cook, Robert L. (1984). „Shade Trees“. In: *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '84. Association for Computing Machinery, S. 223–231. DOI: 10.1145/800031.808602.
- Cook, Robert L., Loren Carpenter und Edwin Catmull (1987). „The Reyes Image Rendering Architecture“. In: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '87. Association for Computing Machinery, S. 95–102. DOI: 10.1145/37401.37414.
- Dachsbaecher, Carsten und Natalya Tatarchuk (2007). *Prism Parallax Occlusion Mapping with Accurate Silhouette Generation*. URL: <https://hal.inria.fr/inria-00606806> (besucht am 30.06.2022).
- Drebin, Robert A., Loren Carpenter und Pat Hanrahan (1988). „Volume Rendering“. In: *SIGGRAPH Comput. Graph.* 22, S. 65–74. DOI: 10.1145/378456.378484.
- El Jamiy, Fatima und Ronald Marsh (Apr. 2019). „A Survey on Depth Perception in Head Mounted Displays: Distance Estimation in Virtual Reality, Augmented Reality and Mixed Reality“. In: *IET Image Processing* 13. DOI: 10.1049/iet-ipr.2018.5920.

- Gateau, Samuel und Steve Nash (20. Sep. 2010). *Implementing Stereoscopic 3D in Your Applications*. NVIDIA. URL:  
[https://www.nvidia.com/content/GTC-2010/pdfs/2010\\_GTC2010.pdf](https://www.nvidia.com/content/GTC-2010/pdfs/2010_GTC2010.pdf) (besucht am 30.06.2022).
- Hart, J. C., D. J. Sandin und L. H. Kauffman (1989). „Ray Tracing Deterministic 3-D Fractals“. In: *SIGGRAPH Comput. Graph.* 23, S. 289–296. DOI: 10.1145/74334.74363.
- Hart, John (1996). „Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces“. In: *The Visual Computer* 12. DOI: 10.1007/s003710050084.
- HP Development Company, L.P (2022). *HP Reverb G2 Virtual Reality Headset Specifications*. URL: <https://support.hp.com/de-de/document/c06938191#AbT2> (besucht am 30.06.2022).
- Kaneko, Tomomichi, Toshiyuki Takahei, Masahiko Inami, Naoki Kawakami, Yasuyuki Yanagida, Taro Maeda und Susumu Tachi (Jan. 2001). „Detailed shape representation with parallax mapping“. In: *In Proceedings of the ICAT 2001*.
- Kufner, Robert (2017). „Quasi volumetrisches Rendering einer gridbasierten 2D Smokesimulation unter Nutzung von Z-buffer und Parallax Mapping“. Masterarb. Institut für Informatik. Ludwig-Maximilians-Universität München.
- Mayerhöfer, Thomas G, Susanne Pahlow und Jürgen Popp (2020). „The Bouguer-Beer-Lambert law: Shining light on the obscure“. In: *ChemPhysChem* 21.18, S. 2029–2046.
- McGrattan, Kevin B und Glenn P Forney (2004). *Fire Dynamics Simulator User's Guide (Sixth Edition)*. Techn. Ber. Version 6. DOI: 10.6028/NIST.SP.1019.
- Reeves, William (1983). „Particle systems—a technique for modeling a class of fuzzy objects“. In: *Proceedings of the 10th annual conference on Computer graphics and interactive techniques - SIGGRAPH '83*. ACM Press, S. 359–375. DOI: 10.1145/800059.801167.
- Schlager, Bettina (2017). „Building a Virtual Reality Fire Training with Unity and HTC Vive“. In: *Proceedings of Central European Seminar on Computer Graphics for students (CESCG 2017)*. URL: <https://www.vrvis.at/publications/PB-VRVis-2017-008>.
- Schuemie, Martijn J, Peter Van Der Straaten, Merel Krijn und Charles APG Van Der Mast (2001). „Research on presence in virtual reality: A survey“. In: *CyberPsychology & Behavior* 4.2, S. 183–201.
- Sutherland, Ivan E. (1965). „The Ultimate Display“. In: *Proceedings of the IFIP Congress*, S. 506–508.
- (1968). „A head-mounted three dimensional display“. In: *AFIPS Conference Proceedings*. Bd. 33. ACM, S. 295–302. DOI: 10.1145/280811.281016.

- Tatarchuk, Natalya (2006). „Practical Parallax Occlusion Mapping with Approximate Soft Shadows for Detailed Surface Rendering“. In: *ACM SIGGRAPH 2006 Courses*. SIGGRAPH '06. Association for Computing Machinery, S. 81–112. DOI: 10.1145/1185657.1185830.
- Tauer, H. (2010). *Stereo-3D: Grundlagen, Technik und Bildgestaltung*. Fachverlag Schiele & Schoen, S. 20–70. ISBN: 9783794907915. URL: <https://books.google.de/books?id=mfJ4HjD6CwEC> (besucht am 26.06.2022).
- Wald, Ingo, Andreas Dietrich, Carsten Benthin, Alexander Efremov, Tim Dahmen, Johannes Gunther, Vlastimil Havran, Hans-peter Seidel und Philipp Slusallek (2006). „Applying Ray Tracing for Virtual Reality and Industrial Design“. In: *2006 IEEE Symposium on Interactive Ray Tracing*, S. 177–185. DOI: 10.1109/RT.2006.280229.
- Welsh, Terry (2004). „Parallax mapping with offset limiting: A perpixel approximation of uneven surfaces“. In: *Infiscape Corporation*.
- Williams-Bell, F. M., B. Kapralos, A. Hogue, B. M. Murphy und E. J. Weckman (Mai 2014). „Using Serious Games and Virtual Simulation for Training in the Fire Service: A Review“. In: *Fire Technology* 51.3, S. 553–584. DOI: 10.1007/s10694-014-0398-1.
- Zhang, Tianli (2020). „FOgggy-LAser Rendering Method for Interaction in Virtual Reality“. Masterarb. Computer, Information Sciences, KTH School of Electrical Engineering und Computer Science. URL: <http://www.diva-portal.org/smash/get/diva2:1420642/FULLTEXT01.pdf> (besucht am 30.06.2021).

## Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorgelegte Abschlussarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

---

Ort, Datum

---

Rechtsverbindliche Unterschrift

TH Köln  
Gustav-Heinemann-Ufer 54  
50968 Köln  
[www.th-koeln.de](http://www.th-koeln.de)

Technology  
Arts Sciences  
**TH Köln**